AI HW3 Report 109550175

1. Report Outline :
   a. 各項實驗指標介紹
   b. 各個實驗內容(都模擬共 100 局做母體)、結果以及給出可能的判讀
   c. 討論 Resolution-based agent 和他種 agent 的優劣
   d. 作業心得以及 Appendix , 附上 link to source code + experiment data
   e. source code along with comment in text(not counted to 10 pages limit)
2. 各項實驗指標介紹 ：

   在這份報告中我自己定義了一些評斷盤面的指標，以下一一介紹 ：

   A. Stuck : 定義何時已經推論不出可以被 marked 的 cell

```
while(Check_Win() == 0){
    iter_cnt += 1 ;
    int prev_num_marked = num_marked ;
    //cout<<"KB.size() is "<<KB.size()<<" ; mean KB size is : "<<MeanKBSize()<<endl;
    if(KB.size() > 15000){
        break ;
    }
    int dilema = GameFlow() ;
    if(dilema == 1 || Check_Logic() == 1 || Check_Statement() == 1){
        cout<<"Wrong Logic !"<<endl;
        break ;
    }
    //cout<<"In "<<iter_cnt<<" 's iteration , Founded Mines : " ;
    //cout<<founded_mines<<endl ;
      ShowMarked() ;
    //cout<<"=============="<<endl;
    if(num_marked == prev_num_marked){
        stuck += 1 ;
        if(KB.size() == prev_KBsize){
            stuck += 3 ;
        }
        cout<<"stuck in cnt : "<<stuck<<" , KB size is "<<KB.size()<<endl;
        if(stuck >= 8){
            break ;
        }
    }
```

   定義如果(1)過了 8 次 iteration 後(看過 KB 8 次後)，仍未能夠有新的 cell 被 mark(2)KB size 太大，超過 15000 個(3)KB size 保持一致超過很多次，此情況很常發生在死局狀態下。以上幾種情況下，都會 break 出推論的 while 迴圈，也就是此篇報告說的 stuck 狀態。

   B. Find : 當一局遊戲終了時，有多少盤面上的 cell 已經被 marked

```
double Find_percentage(){
    int cnt = 0 ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] != 0){
                cnt += 1 ;
            }
        }
    }
//    cout<<cnt<<" / "<<(size1*size2)<<endl;
    return double(cnt) / double(size1*size2) ;
}
```

C. Concentration : 一局遊戲中地雷彼此之間的歐式距離總和，再取平均

```
double MinesAvgDistance(){ // to model the mines' concentration
    assert(WhereMines.size() == mines) ;
    double dist = 0 ;
    for(int i = 0 ; i < WhereMines.size() ; i++){
        for(int j = 0 ; j < WhereMines.size() ; j++){
            if(i != j){
                int x1 = WhereMines[i].first ; int x2 = WhereMines[j].first ;
                int y1 = WhereMines[i].second ; int y2 = WhereMines[j].second ;
                double aa = double((double(x1) - double(x2)) *  (double(x1) - double(x2))) + \
                double((double(y1) - double(y2)) *  (double(y1) - double(y2))) ;
                double DistBtwPairs = sqrt(aa) ;
                dist += DistBtwPairs ;
            }
        }
    }
    double avg_dist = dist / double(2 * WhereMines.size()) ;
    return avg_dist ;
}
```

這項指標是用來描述一盤遊戲中地雷的集中程度。

D. Connection : 一局遊戲中，那些起始提示用的 safe cells 平均可以推論出多少並非起始提示的 cells

```
double InitCellConnection(){
    double cc = 0 ;
    for(int i = 0 ; i < WhereInitHints.size() ; i++){
        int target_r = WhereInitHints[i].first ;
        int target_c = WhereInitHints[i].second ;
        cc += BFS(target_r , target_c) ;
    }
```

```
int BFS(int r , int c){
    queue<pair<int,int> >q ;
    q.push({r,c}) ;
    int connection = -1 ;
    int visited[size1*size2] ;
    for(int i = 0 ; i < size1*size2 ; i++){
        visited[i] = 0 ;
    }
    int iid = r*size2 + c ;
    visited[iid] = 1 ;
    while(!q.empty()){
        pair<int,int>cur = q.front() ;
        q.pop() ;
        int row = cur.first ;
        int col = cur.second ;
        connection += 1 ;
        if(row - 1 >= 0 && col - 1 >= 0 && (marked[row-1][col-1] != 0)){
            int id = (row-1)*size2 + col-1 ;
            if(visited[id] == 0 && In_InitHint(row-1 , col-1) == 0){
                visited[id] = 1 ;
                q.push({row-1 , col-1}) ;
            }
        }
        if(row - 1 >= 0 && col + 1 < size2 && (marked[row-1][col+1] != 0)){
            int id = (row-1)*size2 + col+1 ;
            if(visited[id] == 0 && In_InitHint(row-1 , col+1) == 0){
                visited[id] = 1 ;
                q.push({row-1 , col+1}) ;
            }
        }
        if(row + 1 < size1 && col - 1 >= 0 && (marked[row+1][col-1] != 0)){
            int id = (row+1)*size2 + col-1 ;
            if(visited[id] == 0 && In_InitHint(row+1 , col-1) == 0){
                visited[id] = 1 ;
                q.push({row+1 , col-1}) ;
            }
        }
```

```
        if(row + 1 < size1 && col + 1 < size2 && (marked[row+1][col+1] != 0)){
            int id = (row+1)*size2 + col+1 ;
            if(visited[id] == 0 && In_InitHint(row+1 , col+1) == 0){
                visited[id] = 1 ;
                q.push({row+1 , col+1}) ;
            }
        }
        if(row + 1 < size1 && (marked[row+1][col] != 0)){
            int id = (row+1)*size2 + col ;
            if(visited[id] == 0 && In_InitHint(row+1 , col) == 0){
                visited[id] = 1 ;
                q.push({row+1 , col}) ;
            }
        }
        if(row - 1 >= 0 && (marked[row-1][col] != 0)){
            int id = (row-1)*size2 + col ;
            if(visited[id] == 0 && In_InitHint(row-1 , col) == 0){
                visited[id] = 1 ;
                q.push({row-1 , col}) ;
            }
        }
        if(col + 1 < size2 && (marked[row][col+1] != 0)){
            int id = (row)*size2 + col+1 ;
            if(visited[id] == 0 && In_InitHint(row , col+1) == 0){
                visited[id] = 1 ;
                q.push({row , col+1}) ;
            }
        }
        if(col - 1 >= 0 && (marked[row][col-1] != 0)){
            int id = (row)*size2 + col-1 ;
            if(visited[id] == 0 && In_InitHint(row , col-1) == 0){
                visited[id] = 1 ;
                q.push({row , col-1}) ;
            }
        }
    }
    return connection ;
}
```

```
int In_InitHint(int r , int c){
    for(int i = 0 ; i < WhereInitHints.size() ; i++){
        int rr = WhereInitHints[i].first ;
        int cc = WhereInitHints[i].second ;
        if(rr == r && cc == c){
            return 1 ;
        }
    }
    return 0 ;
}
```

將各起始提示位置的 safe cell 當作 BFS 起點，一層一層向外拓展，直到一個 cell(either 是提示位置的 cell 或是由提示位置拓展到的 cell)的 eight neighbors 裡面都沒有從上一層拓展而來的 cells(提示時 initialze 的那些 safe cell 不能算做滿足條件的 eight neighbors)，最後將其平均回傳。可以用來評估某種 Initiailize safe cell 的方式下這些 safe cell 對盤面推論的貢獻度。

E. Speed: 一局遊戲推論到終局(Find = 100%)或是 stuck 的狀態時總共花了多少”秒”，所以 Speed 越大其實代表推論速度越慢。(這是我命名上的瑕疵)

3. 實驗內容、結果以及對應之判讀 : (除 Exp1 外都以 task3(難度 Hard)做實驗)

Exp1. 不同難度下推論到終局(Find = 100%)或 stuck 的速度 : which is trivial

task1 v.s. task2 v.s. task3

AVG(Max) speed : 0.188868(2.1604) v.s. 0.9382995(1.79391) v.s. 37.1331(716.475)

AVG(Min) Find : 99.87655(95.0617) v.s. 99.78125(96.4844) v.s. 78.729(11.875)

Exp2. 不同 Hint initialization 方式對解題的影響(比較推論到終局或 stuck 的 speed、Find 以及 Connection) :

Serially initialization :

```
vector<vector<string> > Initial_Safe_Cells(int type = -1){ // initialize safe cell hint in a serial manner
    int number_of_safe_hint = -1 ;
    if(type == -1){
        number_of_safe_hint = round(sqrt(size1*size2)) ;
    }
    else{
        number_of_safe_hint = type ;
    }
    vector<vector<string> >container ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(Board[i][j] == 0){
                string literal =  Generate(i , j , 1) ;
                vector<string>clause ;
                clause.push_back(literal) ;
                container.push_back(clause) ; // because it is single length clause , so we don't need to sort this clause before inserting
                pair<int,int>pp(i , j) ;
                WhereInitHints.push_back(pp) ;
            }
            if(container.size() == number_of_safe_hint){
                return container ;
            }
        }
    }
}
```

Randomly initialization :

```
vector<vector<string> > Initial_Safe_Cells_Random(int type = -1){ // initialize safe cell hint in a random manner
    int number_of_safe_hint = -1 ;
    if(type == -1){
        number_of_safe_hint = round(sqrt(size1*size2)) ;
    }
    else{
        number_of_safe_hint = type ;
    }
    vector<int>visited(size1*size2) ;
    for(int i = 0 ; i < visited.size() ; i++){
        visited[i] = 0 ;
    }
    vector<vector<string> >container ;
    while(container.size() < number_of_safe_hint){
        int randNum = rand()%(size1*size2) ;
        int i = randNum/size2 ; int j = randNum%size2 ;
        if(Board[i][j] == 0 && visited[randNum] == 0){
            string literal =  Generate(i , j , 1) ;
            vector<string>clause ;
            clause.push_back(literal) ;
            container.push_back(clause) ;
            pair<int,int>pp(i , j) ;
            WhereInitHints.push_back(pp) ;
            visited[randNum] = 1 ;
        }
    }
    return container ;
}
```

Serial v.s. Random :

AVG speed : 10.761745 v.s. 37.1331

AVG Find : 77.7104 v.s. 78.729

AVG connection : 337.851 v.s. 319.666

Max speed : 188.604 v.s. 716.475

解讀 : (1) 對於盤面整體推論的成效來講，這兩種方式的強度差不多，random 方式小有優勢，原因是 : serial 方式的初始訊息雖然較集中，但僅能將提示附近區

塊的 cells 推論完整而無法擴展到較遠的 cells；random 方式下雖然提示擴及的範圍較廣，但是也只能在提示附近小範圍的 mark，而無法保證將這些散落的 marked 區域串聯起來。下方兩個殘局範例可以很好地描述此現象 ：

某盤以 Serial 方式 initialize 後，推論到 stuck 狀態的最終 marked 結果(0 是 safe cell , 1 是 mine , *是未知(unmarked)):

```
00000*00*00*0*0*1010*000000000
01110*00********1110*100000000
**********************000001000
**********************000100001
**********************011000111
**********************1000110*
**********************1100000*
**********************0000101*
**********************0100000*
**********************0000001*
**********************0001000*
**********************0000000*
****************************
****************************
****************************
****************************
```

某盤以 random 方式 initialize 後，推論到 stuck 狀態的最終 marked 結果

```
00011**********000**********
00001000*********000**********
00000010*******10001*********
10000000*0********************
10011001**0*******0***********
00100000*********************
10000000****0****************
00000000**********0*********
00001011*********************
0000000***0*******************
0011000******0****************
0000001***********0**1*******
0100010********10010000******
000000*********1000100000*****
***************10000000******
***************00000000******
```

(2) 雖然 Find 方面表現得差不多，但是 AVG connection 方面卻是 Serial 明顯比較高。回到上方介紹 connection 這項指標的 code 內容，我們可以看到每個提示 cell 之間計算的 connection 可能會同時涵蓋到盤面上某個新 marked 的 cell，這也恰好證實了 serial 方式的 marked 區塊是集中的、區域性的、極度仰賴 serially initializtion 時所選擇的區塊的。

(3) 速度上，randomly initialization 慢得多，這是因為如果當我們分散地給出提示點，我們每次推進 KB 內部的 CNF clause 不僅通常會包含很多 literals(因為周圍的 cell 都沒被 marked 過)，clauses 之間進行邏輯上的 subsumption 或是 resolution 的頻率也比較低一點，使得 KB.size()有時會很胖，大大減緩程式運行時的執行速度(因為 unmarked cell 之間資訊太過分散，因此遊戲初期一小塊區域內相關的 clauses 之間很難作化簡，僅能依靠看完整個 KB+隨著遊戲進程逐步將 KB 清空)。

Exp3. 不同 Concentration 下對解題的影響：

除了採用 task3 外這裡一律採用 randomly initialize 的方式。

我自己設計了一個實驗，想探討盤面上地雷分布的密集程度對於解題完程度有什麼影響。其中我在跑了前幾個實驗後發現 concentration 的分布多在 500-675 之間，因此我將此區段畫為三等份 ：550+-25 , 600 +-25 以及 650 +-25 分別去做實驗。

550 v.s. 600 v.s. 650

AVG Find : 75.1042 v.s. 79.4396 v.s 84.15625

AVG connection : 290.9435 v.s. 326.578 v.s. 356.869

解讀 ： 我們可以觀察到在 hint 被 randomly initialized 的情況下，地雷分散的情況下可以有較好的解題率，同時每個 initial cell 的 connection 也相對較高。我猜測這是因為 concentration 650+-25 的限制使得地雷分布最接近如同 hint 一般的隨機分布，因此可以成功藉由這個分布定位出有關這些 hint 周圍的地雷位置，又因為我們確切的定位了這些 mines 的位置，我們可繼續拓展將其他 safe cell marked 起來，而不會陷入”不能夠確定哪個 cell 是 mine 哪個 cell 是 safe 的”而 stuck 的窘境。

Exp4. 不同 hint_num 對解題的影響(# initial safe cell 22 v.s. 44 v.s. 66 v.s. 88) :

除了採用 task3 外這裡一律採用 randomly initialize 的方式。

22 v.s. 44 v.s. 66 v.s. 88

AVG Find : 78.729 v.s. 89.79165 v.s. 95.2604 v.s. 96.2417

Min Find : 11.875 v.s. 35.625 v.s. 55.2083 v.s. 78.5417

解讀 :(1) 我們可以看到隨著提示數目的增加，AVG Find 比例甚至兜底的 Min Find 都越來越高，這可以說明 Logical agent 的強度與初始提示資訊量是呈正相關的。這給了我們在抉擇是否選用 logical agent 完成任務時一個很好的指導原則 : 我們對於未知環境不能貿然選用。(2) 我們可以看到 66 跟 88 兩次實驗下，AVG Find 的成長比例不高，仍有 30*16*0.04 約 20 個 cell 推論不出來，這說明了 Logical agent 也有其極限: 當環境存在多種可能時(如同那 20 個 cell 可能有多個 mine，safe cells 的分布，且都滿足邏輯正確性)時，logical agent 可能需要他種方式的輔助如枚舉所有可能+評估各個可能的對應分數值去做出最佳的下一步動作等。

4. 分析與討論 Resolution-based agent 和他種方式的 agent 的優劣性 : 在完成程式碼的同時，我開始思考這種純粹用邏輯推論完成遊戲的 game agent 的優劣勢在哪，乃至拓展到不同任務上(不僅限於踩地雷)，各種推論方法的 agent 該如何選擇。以下是我的統整分析：

   A. Pure Logic Based agent :

      優勢 :

      (1) 推論一定是 sound 的，意即這種 agent 一定不會犯錯，靠的就是基本的邏輯推論保證正確性。

      (2) 高效性，因為 Pure Logic Based 的 agent 如 resolution-based 的 agent 僅考慮與解決方案相關的子集，因此在資訊量相同的情況下這種 agent 傾向於更快收束到問題答案。

      劣勢 :

      (1) 易受搜索空間大小的限制，例如這次的踩地雷 agent，如果採用很隨機的 initialize safe cells 的方式，很容易有提示 cell 周圍都沒有確定 cell 的狀況會出現，這時我們就會有很多且長度很長的 clause 必須推進 KB 內做 resolution 或是簡化(check subsumption)，當 KB size 一大，運氣不好的情況下就要推論很久。

      (2) 可能陷入死循環(stuck)，如果我們給 agent 的資訊不夠完備，agent 會無法確切證明某個命題是否為真因而原地空轉，這可能就會導致運算資源的浪費。此外這也意味著此種 agent 探索未知的能力極度受到初始條件的限制。在實作踩地雷 agent 的時候，這種情況很常在 task3(Hard) 出現。

      ➔因此，這種 agent 只能運用在較小型的問題上，對於大型的、未知環境的問題則不適合。且此問題必須可以清楚的定義某個條件的真值。

B. Heurisitic Based agent :

優勢 ：

(1) 效率高：此種 agent 可以使用啟發函數來指導搜索解的過程，可以幫助 agent 更快地找到解決方案，因為啟發函數的使用通常可以避免搜索所有可能的解決方案，這使得它在大型問題上比 Logic-Based 的 agent 更加高效。

(2) 可解釋性佳並且對於已經掌握足夠知識的特定問題，解題效果顯著。

劣勢 ：

(1) 在設計啟發函數時需要 domain knowledge，我們很難對一個陌生的環境使用 heuristic based 的 agent，因為我們無法定義有效的評分標準。

(2) 可能收束到的解法並非最佳(因為並沒有搜索所有可能的解決方法)，而是落在 local extremum，而啟發函數的設計品質與這現象息息相關。

➔因此，這種 agent 適合在我們熟悉的環境下，針對不同任務做快速推論。

C. Reinforcement Based agent :

優勢 ：

(1) 不需要 domain knowledge，也不需要特別設計啟發函數作為 agent 動作的依據。

(2) 極高的適應及擴展性 ：因為 Reinforcement Learning 的核心概念就是與環境做互動，並且藉由反饋進行學習，所以此觀念在我們面對不同甚至是未知的環境時可以多方沿用。

劣勢 ：

(1) 一個 agent 需要和環境進行足夠多的互動才可以做出合理的動作判斷。這也代表著時間以及資源的消耗在所難免，此外其狀態空間以及行動空間也不能夠太大。反之，如果我們以過少的互動經驗或是噪聲太多的環境作為判斷依據，agent 很可能會出現 overfitting 的狀況而無法做出最優判斷。

(2) 需考慮 exploitation 和 exploration 之間的 trade-off

(3) 有些任務是無法進行試誤學習的 ：reinforcement learning 強調的就是從經驗中學習，但如果一個毫無經驗且沒有任何參考訓練資料的 agent 要學習開自駕車、踩地雷等等任務時，我們很有可能在成功學習前就已經發生事故、踩到地雷而輸掉遊戲了。

➔因此，這種 agent 適合運用在探索未知的環境上，並且任務本身對於 agent 的學習速度並未有嚴苛要求。此外仍需考慮 agent 與環境互動的過程中所需的試誤成本(需要有相對應的模擬環境讓 agent 學習，以解決此類限制)

5. 心得 ：

這是我第一次著手包辦一個完整的 agent 程式，之前在 Introduction to AI 時寫的 Pac Man 可以動手做的部分只有自定義 scoring function 的部分，以及將一些給定好的理論如 Q-learning、SARSA 填入適當的空格內，因此我很喜歡這次的作業內容,同時此次作業也很好的呼應我們上課時所教授的 logical agent 的相關內容。覺得最印象深刻的部分除了要從 1000 多行 C++ code 挑燈夜戰找 bug 外，我還認知到了"不讓 logical agent 的 Knowledge Base 多到無法負荷"其實是此類 agent 的難點：要怎麼樣維持 sound and complete 的推論同時又 maintain KB 的簡潔以避免做出回應的時長過長是項重要的課題。

6. Links to experiment data and source code :

Data :

https://drive.google.com/drive/folders/1gncQEIkMr4akMTt52gN4szxnJIRQxXEN?usp=sharing

Source Code :

https://drive.google.com/drive/folders/1UiD2DjOMrapg3-AtD4DZjnVR_pMyilLe?usp=sharing

p.s. :

(我強烈建議教授或助教們批改時下載 code，並以可排版、收放函數內容的 editor 開啟，因為共一千多行，我已經盡量寫得很乾淨但還是很多行，非常抱歉)

下頁開始為 code 的純文字檔，以 nodepad++剪貼文字以保證 code 排版不變，請見諒。

(方法如下 ：https://officeguide.cc/word-insert-code-preserving-format-and-syntax-highlighting-tutorial-examples/)

```
#include <bits/stdc++.h>
#include <sys/time.h>
#include <iostream>
#include <fstream>
using namespace std;
int Board[16][30] ;
int marked[16][30] ; // 0 as unmarked , 1 marked as mine pos , 2 marked as
safe pos
vector<pair<int,int> >WhereMines ; // after each time a board is
initialized , the location of mines are stored here
vector<pair<int,int> >WhereInitHints ;  // after each time a board is
initialized , the location of initial hints are stored here
int difficulty = -1 ; // 1 as easy , 2 as medium , 3 as hard
int init_type = -1 ; // 1 as serially initialize the hint ; 2 as randomly
initialize the hint
int size1 = -1 ; // # row of the board
int size2 = -1 ; // # column of the board
int mines = -1 ; // # mines in the board
int founded_mines = 0 ; // In current game flow , how many mines are already
found
int num_marked = 0 ; // In current game flow , how many cell are already
marked as either safe or mined
vector<vector<string> >KB ; // assume each clause is either "a single-
lateral" or "multi-laterals" in CNF
double MinesAvgDistance() ; // just a declaration of function
int Check_all_global() ;  // just a declaration of function

////////////// Global parameters ////////////////

struct cmp{ // sort literals in a class in their ascending abs() way
   bool operator()(string a , string b){
      if(a[0] == '-'){
         a = a.substr(1) ;
      }
      if(b[0] == '-'){
         b = b.substr(1) ;
      }
      int a_num = atoi(a.c_str()) ;
```

```
        int b_num = atoi(b.c_str()) ;
        return a_num < b_num ;
    }
};
struct cmpp{ // sort clauses in KB in their ascending length way
    bool operator()(vector<string> aa , vector<string> bb){
        return aa.size() < bb.size() ;
    }
};
void Show(){ // Show the Board
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            cout<<Board[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
}
void ShowInitHint(){ // Show where is the hint
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            int flag = 0 ;
            for(int k = 0 ; k < WhereInitHints.size() ; k++){
                int r = WhereInitHints[k].first ;
                int c = WhereInitHints[k].second ;
                if(r == i && c == j){
                    flag = 1 ;
                    break ;
                }
            }
            if(flag == 0){
                cout<<"*"<<" ";
            }
            else{
                cout<<0<<" ";
            }
        }
        cout<<endl;
```

```
    }
    cout<<endl;
}
void ShowKB(){ // Show clauses in KB
    sort(KB.begin() , KB.end() , cmpp()) ;
    for(int i = 0 ; i < KB.size() ; i++){
        for(int j = 0 ; j < KB[i].size() ; j++){
            cout<<KB[i][j]<<" " ;
        }
        cout<<endl;
    }
    cout<<endl;
}
void ShowMarked(){ // Show marked map
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] == 1){
                cout<<1<<" ";
            }
            else if(marked[i][j] == 2){
                cout<<0<<" ";
            }
            else{ // unmarked(unknown) cell
                cout<<"*"<<" ";
            }
        }
        cout<<endl ;
    }
    cout<<endl;
}

//////////////  Utility Functions  ///////////////

void Initialize(int number_of_mines = -1 , int concentration = -1){ //
Initialize the Board when an epoch starts
    vector<pair<int,int> >WhereMines_replicate ;
    WhereMines = WhereMines_replicate ;
    vector<pair<int,int> >WhereInitHints_replicate ;
```

```
WhereInitHints = WhereInitHints_replicate ;
founded_mines = 0 ;
num_marked = 0 ;
vector<vector<string> >KB_flush ;
KB = KB_flush ;
if(difficulty == 1){
    size1 = 9 ;
    size2 = 9 ;
    mines = 10 ;
}
else if(difficulty == 2){
    size1 = 16 ;
    size2 = 16 ;
    mines = 25 ;
}
else if(difficulty == 3){
    size1 = 16 ;
    size2 = 30 ;
    mines = 99 ;
}
else{
    size1 = 16 ;
    size2 = 30 ;
    mines = number_of_mines ;
}
for(int i = 0 ; i < size1 ; i++){
    for(int j = 0 ; j < size2 ; j++){
        Board[i][j] = 0 ;
        marked[i][j] = 0 ;
    }
}
assert(Check_all_global() == 0) ;
if(difficulty <= 3){
    int cnt = 0 ;
    while(cnt < mines){
        int randNum = rand()%(size1*size2) ;
        int row = randNum/size2 ;
        int col = randNum%size2 ;
```

```
            if(Board[row][col] == 0){
                Board[row][col] = 1 ;
                cnt += 1 ;
                pair<int,int>coord(row , col) ;
                WhereMines.push_back(coord) ;
            }
        }
    }
    else{ // self defined hard
        while(1){ // Keep initializing , till the board matches the
concentration requirements
            int cnt = 0 ;
            while(cnt < mines){
                int randNum = rand()%(size1*size2) ;
                int row = randNum/size2 ;
                int col = randNum%size2 ;
                if(Board[row][col] == 0){
                    Board[row][col] = 1 ;
                    cnt += 1 ;
                    pair<int,int>coord(row , col) ;
                    WhereMines.push_back(coord) ;
                }
            }
            double res = MinesAvgDistance() ;
            if((concentration == -1) || (res >= concentration - 25 && res <=
concentration + 25)){
                break ;
            }
            else{
                for(int i = 0 ; i < size1 ; i++){
                    for(int j = 0 ; j < size2 ; j++){
                        Board[i][j] = 0 ;
                    }
                }
                vector<pair<int,int> >WhereMines_replicate_ ;
                WhereMines = WhereMines_replicate_ ;
            }
        }
```

```
    }

}
pair<vector<int> , int> Hint(int row , int col){ // generate hint , which
contains m un-marked neighbors and n mines in there
    int cnt = 0 ;
    vector<int>unmarked_neighbors ;
    if(row - 1 >= 0 && col - 1 >= 0 && (marked[row-1][col-1] == 0)){
        unmarked_neighbors.push_back((row-1)*size2 + col-1) ;
        if(Board[row-1][col-1] == 1){
            cnt += 1 ;
        }
    }
    if(row - 1 >= 0 && col + 1 < size2 && (marked[row-1][col+1] == 0)){
        unmarked_neighbors.push_back((row-1)*size2 + col+1) ;
        if(Board[row-1][col+1] == 1){
            cnt += 1 ;
        }
    }
    if(row + 1 < size1 && col - 1 >= 0 && (marked[row+1][col-1] == 0)){
        unmarked_neighbors.push_back((row+1)*size2 + col-1) ;
        if(Board[row+1][col-1] == 1){
            cnt += 1 ;
        }
    }
    if(row + 1 < size1 && col + 1 < size2 && (marked[row+1][col+1] == 0)){
        unmarked_neighbors.push_back((row+1)*size2 + col+1) ;
        if(Board[row+1][col+1] == 1){
            cnt += 1 ;
        }
    }
    if(row + 1 < size1 && (marked[row+1][col] == 0)){
        unmarked_neighbors.push_back((row+1)*size2 + col) ;
        if(Board[row+1][col] == 1){
            cnt += 1 ;
        }
    }
    if(row - 1 >= 0 && (marked[row-1][col] == 0)){
```

```
        unmarked_neighbors.push_back((row-1)*size2 + col) ;
        if(Board[row-1][col] == 1){
            cnt += 1 ;
        }
    }
    if(col + 1 < size2 && (marked[row][col+1] == 0)){
        unmarked_neighbors.push_back((row)*size2 + col+1) ;
        if(Board[row][col+1] == 1){
            cnt += 1 ;
        }
    }
    if(col - 1 >= 0 && (marked[row][col-1] == 0)){
        unmarked_neighbors.push_back((row)*size2 + col-1) ;
        if(Board[row][col-1] == 1){
            cnt += 1 ;
        }
    }
    sort(unmarked_neighbors.begin() , unmarked_neighbors.end()) ;
    return pair<vector<int> , int>(unmarked_neighbors , cnt) ;
}
string Generate(int row , int col , int negate , int num_id = -1){ //
generate literal in string format
    string str_id = "" ;
    stringstream ss ;
    if(num_id == -1){
        int id = row * size2 + col ;
        ss << (id+1) ; // to handle 0 , which is hard to find diff between
negated literal or non-negated one ;
        ss >> str_id ;
    }
    else{
        ss << (num_id+1) ;
        ss >> str_id ;
    }

    if(negate == 1){
        return "-" + str_id ;
    }
```

```
    else{
        return str_id ;
    }
}
vector<vector<string> > Initial_Safe_Cells(int type = -1){ // initialize
safe cell hint in a serial manner
    int number_of_safe_hint = -1 ;
    if(type == -1){
        number_of_safe_hint = round(sqrt(size1*size2)) ;
    }
    else{
        number_of_safe_hint = type ;
    }
    vector<vector<string> >container ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(Board[i][j] == 0){
                string literal =  Generate(i , j , 1) ;
                vector<string>clause ;
                clause.push_back(literal) ;
                container.push_back(clause) ; // because it is single length
clause , so we don't need to sort this clause before inserting
                pair<int,int>pp(i , j) ;
                WhereInitHints.push_back(pp) ;
            }
            if(container.size() == number_of_safe_hint){
                return container ;
            }
        }
    }
}
vector<vector<string> > Initial_Safe_Cells_Random(int type = -1){ //
initialize safe cell hint in a random manner
    int number_of_safe_hint = -1 ;
    if(type == -1){
        number_of_safe_hint = round(sqrt(size1*size2)) ;
    }
    else{
```

```
        number_of_safe_hint = type ;
    }
    vector<int>visited(size1*size2) ;
    for(int i = 0 ; i < visited.size() ; i++){
        visited[i] = 0 ;
    }
    vector<vector<string> >container ;
    while(container.size() < number_of_safe_hint){
        int randNum = rand()%(size1*size2) ;
        int i = randNum/size2 ; int j = randNum%size2 ;
        if(Board[i][j] == 0 && visited[randNum] == 0){
            string literal =  Generate(i , j , 1) ;
            vector<string>clause ;
            clause.push_back(literal) ;
            container.push_back(clause) ;
            pair<int,int>pp(i , j) ;
            WhereInitHints.push_back(pp) ;
            visited[randNum] = 1 ;
        }
    }
    return container ;
}

////////////// Game Module Functions //////////////

int Search_Subliteral(vector<string>clause , string target_literal){ //
search a literal in a clause , assume the clause is ordered , so use BS
    int res = 0 ; // not found
    int l = 0 ; int r = clause.size()-1 ;
    while(l <= r){
        int mid = (l+r)/2 ;
        if(abs(atoi(clause[mid].c_str())) ==
abs(atoi(target_literal.c_str()))){
            if(atoi(clause[mid].c_str()) != atoi(target_literal.c_str())){
                res = -1 ; // ex : -10 v.s. 10
            }
            else{
                res = 1 ; //ex : 3 v.s. 3
```

```
            }
            break ;
        }
        else if(abs(atoi(clause[mid].c_str())) >
abs(atoi(target_literal.c_str()))){ // assume each clause is sort by its abs
literals
            r = mid - 1 ;
        }
        else if(abs(atoi(clause[mid].c_str())) <
abs(atoi(target_literal.c_str()))){
            l = mid + 1 ;
        }
    }
    return res ;
}
void getCombinations(vector<int>& nums, vector<vector<int> >& res , int k) {
    int n = nums.size();
    vector<int> combination(k, 0);
    for (int i = 0; i < k; i++) {
        combination[i] = i;
    }
    while (combination[0] <= n - k) {
        vector<int> curCombination;
        for (int i = 0; i < k; i++) {
            curCombination.push_back(nums[combination[i]]);
        }
        res.push_back(curCombination);
        int t = k - 1;
        while (t != 0 && combination[t] == n - k + t) {
            t--;
        }
        combination[t]++;
        for (int i = t + 1; i < k; i++) {
            combination[i] = combination[i - 1] + 1;
        }
    }
    return ;
}
```

```
vector<vector<int> > Combinations(vector<int>candidates , int k){ //
parameter data type syncs with "Hint()" function
    vector<vector<int> >solutions ;
    getCombinations(candidates , solutions , k) ;
    return solutions ;
}
vector<vector<string> > GenerateClausesFromHint(pair<vector<int> , int>
hint){ // as the function name says
    vector<int>unmarked_neighbors = hint.first ;
    int m = unmarked_neighbors.size() ;
    int n = hint.second ;
    vector<vector<string> >hint_clause ;
    if(n == 0){
        for(int i = 0 ; i < unmarked_neighbors.size() ; i++){
            int num_id = unmarked_neighbors[i] ;
            vector<string>clause ;
            clause.push_back(Generate(-1 , -1 , 1 , num_id)) ; // because
num_id is given , so row , col can pass in any number
            sort(clause.begin() , clause.end() , cmp()) ;
            hint_clause.push_back(clause) ;
        }
    }
    else if(n == m){
        for(int i = 0 ; i < unmarked_neighbors.size() ; i++){
            int num_id = unmarked_neighbors[i] ;
            vector<string>clause ;
            clause.push_back(Generate(-1 , -1 , 0 , num_id)) ;
            sort(clause.begin() , clause.end() , cmp()) ;
            hint_clause.push_back(clause) ;
        }
    }
    else if(m > n && n > 0){
        vector<vector<int> >combs = Combinations(unmarked_neighbors , m-n+1) ;
        for(int i = 0 ; i < combs.size() ; i++){
            vector<string>clause ;
            for(int j = 0 ; j < combs[i].size() ; j++){
                int num_id = combs[i][j] ;
                clause.push_back(Generate(-1 , -1 , 0 , num_id)) ;
```

```
            }
            sort(clause.begin() , clause.end() , cmp()) ;
            hint_clause.push_back(clause) ;
        }

        combs = Combinations(unmarked_neighbors , n+1) ;
        for(int i = 0 ; i < combs.size() ; i++){
            vector<string>clause ;
            for(int j = 0 ; j < combs[i].size() ; j++){
                int num_id = combs[i][j] ;
                clause.push_back(Generate(-1 , -1 , 1 , num_id)) ;
            }
            sort(clause.begin() , clause.end() , cmp()) ;
            hint_clause.push_back(clause) ;
        }
    }
    return hint_clause ;
}
int stricter(vector<string>new_clause , vector<string>old_clause){ // return
which clause is stricter
    int pass1 = 1 ;
    for(int i = 0 ; i < new_clause.size() ; i++){
        string literal = new_clause[i] ;
        if(Search_Subliteral(old_clause , literal) != 1){ // check if all
literals in new_clause is in old_clause
            pass1 = 0 ;
            break ;
        }
    }

    int pass2 = 1 ;
    for(int i = 0 ; i < old_clause.size() ; i++){
        string literal = old_clause[i] ;
        if(Search_Subliteral(new_clause , literal) != 1){ // check if all
literals in old_clause is in new_clause
            pass2 = 0 ;
            break ;
        }
```

```
    }
    if(pass1 == 0 && pass2 == 0){ // 2 clause are not the same
        return -1 ;
    }
    else if(pass1 == 1 && pass2 == 1){ //2 clause are identical
        return 0 ;
    }
    else if(pass1 == 1 && pass2 == 0){ // all literals in new_clause is in
old_clause , but not otherwise , so new one is stricter
        return 1 ;
    }
    else if(pass1 == 0 && pass2 == 1){ // all literals in old_clause is in
new_clause , but not otherwise , so old one is stricter
        return 2 ;
    }
}
pair<vector<int> , int> CheckSubsumption(vector<string>new_clause){ // check
if a concept or clause is already included in KB
    int is_subsumed = 0 ; // case of res == -1 is defaultly handled here
    vector<int>subsumed_by_new ;
    for(int i = 0 ; i < KB.size() ; i++){
        vector<string>old_clause = KB[i];
        int res = stricter(new_clause , old_clause) ;
        if(res == 0){ // 2 clause are identical , new concept is already in KB
==> no need to insert this new clause
            is_subsumed = 1 ;
        }
        else if(res == 1){ // new clause is stricter , we need to delete the
older clause and insert this new clause to not lose the concept
            subsumed_by_new.push_back(i) ;
        }
        else if(res == 2){ // old clause is stricter , new concept is already
in KB ==> no need to insert this new clause
            is_subsumed = 1 ;
        }
    }
    pair<vector<int> , int>info(subsumed_by_new , is_subsumed);
    return info ;
```

```
}
vector<string> DoResolution(vector<string>clause1 , vector<string>clause2 ,
string target_literal_in_clause1){
    string complimentary_literal = "" ;
    if(target_literal_in_clause1[0] == '-'){
        complimentary_literal = target_literal_in_clause1.substr(1) ;
    }
    else{
        complimentary_literal = "-" + target_literal_in_clause1 ;
    }

    vector<string>res1 ;
    for(int i = 0 ; i < clause1.size() ; i++){
        if(clause1[i] != target_literal_in_clause1){
            res1.push_back(clause1[i]) ;
        }
    }

//    sort(res1.begin() , res1.end() , cmp()) ;

    vector<string>res2 ;
    for(int i = 0 ; i < clause2.size() ; i++){
        int flag = 0 ;
        for(int j = 0 ; j < res1.size() ; j++){ // if res1 is already contain
this literal , not append it to avoid like -3 V -3 V 10 in a clause
            if(res1[j] == clause2[i]){
                flag = 1 ;
                break ;
            }
        }
        if(flag == 0 && clause2[i] != complimentary_literal){
            res2.push_back(clause2[i]) ;
        }
    }

    vector<string>result ;
    for(int i = 0 ; i < res1.size() ; i++){
        result.push_back(res1[i]) ;
```

```
    }
    for(int i = 0 ; i < res2.size() ; i++){
        result.push_back(res2[i]) ;
    }
    sort(result.begin() , result.end() , cmp()) ;
    return result ;
}
vector<string> Matching(vector<string>clause1 , vector<string>clause2){ //
assume clause1 , clause2 are not subsume to each other , check if they can
do resolution
    int cnt = 0 ;
    string target_literal_in_clause1 = "";
    for(int i = 0 ; i < clause1.size() ; i++){
        if(Search_Subliteral(clause2 , clause1[i]) == -1){ // contains a piar
of complimentary literal
            cnt += 1 ;
            target_literal_in_clause1 = clause1[i];
            if(cnt > 1){ // If 2 clauses have only one pair of complimentary
literals , then we do resolution , else we do nothing
                break ;
            }
        }
    }

    vector<string> result ;
    if(cnt == 1){
        result = DoResolution(clause1 , clause2 , target_literal_in_clause1) ;
        return result ;
    }
    result.clear() ; // make sure to return an empty vector
    return result ;
}
void Insert_to_KB(vector<int>& need_delete , vector<vector<string> >&
to_append){
    vector<int>need_append(to_append.size()) ;
    for(int i = 0 ; i < need_append.size() ; i++){
        need_append[i] = 0 ;
    }
```

```
    for(int i = 0 ; i < to_append.size() ; i++){
        vector<string>new_clause = to_append[i] ;
        pair<vector<int> , int> cmp_ = CheckSubsumption(new_clause) ;
        if(cmp_.second == 0){ // if new clause's concept is not included in
KB , or new clause is stricter than some clauses in KB
            for(int j = 0 ; j < cmp_.first.size() ; j++){
                int target = cmp_.first[j] ;
                need_delete[target] = 1 ;
            }
            need_append[i] = 1 ;
        }
    }
    vector<vector<string> >KB_replicate ;
    for(int i = 0 ; i < KB.size() ; i++){
        if(need_delete[i] != 1){
            KB_replicate.push_back(KB[i]) ;
        }
    }
    for(int i = 0 ; i < to_append.size() ; i++){
        if(need_append[i] == 1){
            KB_replicate.push_back(to_append[i]) ;
        }
    }
    KB = KB_replicate ;
    sort(KB.begin() , KB.end() , cmpp()) ;
}
int GameFlow(){
    sort(KB.begin() , KB.end() , cmpp()) ; // to ensure finding single-
literal clause first

    vector<int>need_delete(KB.size()) ;
    for(int i = 0 ; i < need_delete.size() ; i++){
        need_delete[i] = 0 ;
    }
    vector<vector<string> >to_append ;

    if(KB[0].size() == 1){ // If there is a single-lateral clause in the KB
        // Mark that cell as safe or mined
```

```
        int id = abs(atoi(KB[0][0].c_str())) - 1 ;
        int row = id/size2 ; int col = id%size2 ;
        if((marked[row][col] == 1 && KB[0][0][0] == '-') || (marked[row][col]
== 2 && KB[0][0][0] != '-')){ // error occurs
            return 1 ;
        }
        int safe_flag = 0 ;
        if(atoi(KB[0][0].c_str()) == (id+1) && marked[row][col] == 0){ //
positive cell , which means there is a mine
            marked[row][col] = 1 ;
            founded_mines += 1 ;
            num_marked += 1 ;
        }
        else if(atoi(KB[0][0].c_str()) != (id+1) && marked[row][col] == 0){
            marked[row][col] = 2 ;
            safe_flag = 1 ;
            num_marked += 1 ;
        }

        // move that clause to KB0 and delete it from KB
        need_delete[0] = 1 ;

        //  Process the "matching" of that clause to all the remaining clauses
in the KB.
        // If new clauses are generated due to resolution, insert them into
the KB.

        for(int i = 1 ; i < KB.size() ; i++){
            vector<string>clause_inKB = KB[i] ;
            vector<string>resolution_res = Matching(KB[0] , clause_inKB) ;
            if(resolution_res.size() > 0){
                to_append.push_back(resolution_res) ;
            }
        }

        // If this cell is safe: Query the game control module for the hint at
that cell.
```

```
        // Insert the clauses regarding its unmarked neighbors into the KB.

        if(safe_flag == 1){
            pair<vector<int> , int>hint = Hint(row , col) ;
            vector<vector<string> >hint_clauses =
GenerateClausesFromHint(hint) ;
            for(int i = 0 ; i < hint_clauses.size() ; i++){
                to_append.push_back(hint_clauses[i]) ;
            }
        }
    }
    else{
        for(int i = 0 ; i < KB.size() ; i++){
            for(int j = 0 ; j < KB.size() ; j++){
                if(i != j && i < j && (KB[i].size() == 2 || KB[j].size() ==
2)){ // only match clause pairs where one clause has only two literals
                    vector<string>resolution_res = Matching(KB[i] , KB[j]) ;
                    if(resolution_res.size() > 0){
                        to_append.push_back(resolution_res) ;
                    }
                }
            }
        }
    }
    Insert_to_KB(need_delete , to_append)  ;
    return 0;
}

////////////   Entire GameFlow   ////////////////

int Check_Logic(){ // Whenever we marked cells , check if it complies with
truth
    int wrong_flag = 0 ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] == 2 && Board[i][j] == 1){
                wrong_flag = 1 ;
                return wrong_flag ;
```

```
            }
            if(marked[i][j] == 1 && Board[i][j] == 0){
                wrong_flag = 1 ;
                return wrong_flag ;
            }
        }
    }
    return wrong_flag ;
}
int Check_Statement(){ // Whenever we generate clauses , check if it
complies with truth
    int wrong_flag = 0 ;
    for(int i = 0 ; i < KB.size() ; i++){
        int cnt = 0 ;
        for(int j = 0 ; j < KB[i].size() ; j++){
            string literal = KB[i][j] ;
            int id = abs(atoi(literal.c_str())) - 1 ;
            int row = id/size2 ; int col = id%size2 ;
            if(literal[0] == '-' && Board[row][col] == 0){
                cnt += 1 ;
            }
            else if(literal[0] != '-' && Board[row][col] == 1){
                cnt += 1 ;
            }
        }
        if(cnt == 0){ // in CNF , at least one literal must be true
            wrong_flag = 1 ;
            break ;
        }
    }
    return wrong_flag ;
}

////////////   Check Logic Functions   ////////////////

int Check_Win(){
    int cnt = 0 ;
    int slot = -1 ;
```

```
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] == 0){
                cnt += 1 ;
                slot = i*size2 + j ;
            }
        }
    }
    if(founded_mines == mines){
        for(int i = 0 ; i < size1 ; i++){
            for(int j = 0 ; j < size2 ; j++){
                if(marked[i][j] == 0){
                    marked[i][j] = 2 ;
                }
            }
        }
        return 1 ;
    }
    if(cnt == 1){
        int row = slot/size2 ; int col = slot%size2 ;
        marked[row][col] = 1 ;
        founded_mines += 1 ;
        return 1 ;
    }
    return 0 ;
}

////////////// Check Win Conditions ////////////////

bool Check_correctness(){
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] == 1 && Board[i][j] == 0){
                return false ;
            }
            if(marked[i][j] == 2 && Board[i][j] == 1){
                return false ;
            }
```

```
        }
    }
    return true ;
}

////////// Check Final Correctness ////////////
double Find_percentage(){
    int cnt = 0 ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(marked[i][j] != 0){
                cnt += 1 ;
            }
        }
    }
//    cout<<cnt<<" / "<<(size1*size2)<<endl;
    return double(cnt) / double(size1*size2) ;
}
int MeanKBSize(){
    int cnt = 0 ;
    for(int i = 0 ; i < KB.size() ; i++){
        cnt += KB[i].size() ;
    }
    return cnt/KB.size()  ;
}
double MinesAvgDistance(){ // to model the mines' concentration
    assert(WhereMines.size() == mines) ;
    double dist = 0 ;
    for(int i = 0 ; i < WhereMines.size() ; i++){
        for(int j = 0 ; j < WhereMines.size() ; j++){
            if(i != j){
                int x1 = WhereMines[i].first ; int x2 = WhereMines[j].first ;
                int y1 = WhereMines[i].second ; int y2 = WhereMines[j].second ;
                double aa = double((double(x1) - double(x2)) *  (double(x1) -
double(x2))) + \
                double((double(y1) - double(y2)) *  (double(y1) -
double(y2))) ;
                double DistBtwPairs = sqrt(aa) ;
```

```
                dist += DistBtwPairs ;
            }
        }
    }
    double avg_dist = dist / double(2 * WhereMines.size()) ;
    return avg_dist ;
}
int In_InitHint(int r , int c){
    for(int i = 0 ; i < WhereInitHints.size() ; i++){
        int rr = WhereInitHints[i].first ;
        int cc = WhereInitHints[i].second ;
        if(rr == r && cc == c){
            return 1 ;
        }
    }
    return 0 ;
}
int BFS(int r , int c){
    queue<pair<int,int> >q ;
    q.push({r,c}) ;
    int connection = -1 ;
    int visited[size1*size2] ;
    for(int i = 0 ; i < size1*size2 ; i++){
        visited[i] = 0 ;
    }
    int iid = r*size2 + c ;
    visited[iid] = 1 ;
    while(!q.empty()){
        pair<int,int>cur = q.front() ;
        q.pop() ;
        int row = cur.first ;
        int col = cur.second ;
        connection += 1 ;
        if(row - 1 >= 0 && col - 1 >= 0 && (marked[row-1][col-1] != 0)){
            int id = (row-1)*size2 + col-1 ;
            if(visited[id] == 0 && In_InitHint(row-1 , col-1) == 0){
                visited[id] = 1 ;
                q.push({row-1 , col-1}) ;
```

```
    }
}
if(row - 1 >= 0 && col + 1 < size2 && (marked[row-1][col+1] != 0)){
    int id = (row-1)*size2 + col+1 ;
    if(visited[id] == 0 && In_InitHint(row-1 , col+1) == 0){
        visited[id] = 1 ;
        q.push({row-1 , col+1}) ;
    }
}
if(row + 1 < size1 && col - 1 >= 0 && (marked[row+1][col-1] != 0)){
    int id = (row+1)*size2 + col-1 ;
    if(visited[id] == 0 && In_InitHint(row+1 , col-1) == 0){
        visited[id] = 1 ;
        q.push({row+1 , col-1}) ;
    }
}
if(row + 1 < size1 && col + 1 < size2 && (marked[row+1][col+1] != 0)){
    int id = (row+1)*size2 + col+1 ;
    if(visited[id] == 0 && In_InitHint(row+1 , col+1) == 0){
        visited[id] = 1 ;
        q.push({row+1 , col+1}) ;
    }
}
if(row + 1 < size1 && (marked[row+1][col] != 0)){
    int id = (row+1)*size2 + col ;
    if(visited[id] == 0 && In_InitHint(row+1 , col) == 0){
        visited[id] = 1 ;
        q.push({row+1 , col}) ;
    }
}
if(row - 1 >= 0 && (marked[row-1][col] != 0)){
    int id = (row-1)*size2 + col ;
    if(visited[id] == 0 && In_InitHint(row-1 , col) == 0){
        visited[id] = 1 ;
        q.push({row-1 , col}) ;
    }
}
if(col + 1 < size2 && (marked[row][col+1] != 0)){
```

```
            int id = (row)*size2 + col+1 ;
            if(visited[id] == 0 && In_InitHint(row , col+1) == 0){
                visited[id] = 1 ;
                q.push({row , col+1}) ;
            }
        }
        if(col - 1 >= 0 && (marked[row][col-1] != 0)){
            int id = (row)*size2 + col-1 ;
            if(visited[id] == 0 && In_InitHint(row , col-1) == 0){
                visited[id] = 1 ;
                q.push({row , col-1}) ;
            }
        }
    }
    return connection ;
}
double InitCellConnection(){
    double cc = 0 ;
    for(int i = 0 ; i < WhereInitHints.size() ; i++){
        int target_r = WhereInitHints[i].first ;
        int target_c = WhereInitHints[i].second ;
        cc += BFS(target_r , target_c) ;
    }
    return double(double(cc)/double(WhereInitHints.size()));
}

////////////  Exp Details    ///////////////////
int Check_all_global(){
    int flag = 0 ;
    for(int i = 0 ; i < size1 ; i++){
        for(int j = 0 ; j < size2 ; j++){
            if(Board[i][j] != 0){
                cout<<"Case1"<<endl;
                flag = 1 ;
            }
             if(marked[i][j] != 0){
                cout<<"Case2"<<endl;
                flag = 1 ;
```

```
            }
        }
    }
    if(WhereMines.size() != 0){
        cout<<"Case3"<<endl;
        flag = 1 ;
    }
    if(WhereInitHints.size() != 0){
        cout<<"Case4"<<endl;
        flag = 1 ;
    }
    if(founded_mines != 0 || num_marked != 0){
        cout<<"Case5"<<endl;
        flag = 1 ;
    }
    if(KB.size() != 0){
        cout<<"Case6"<<endl;
        flag = 1 ;
    }
    return flag ;
}

////////////  Check correctly initialized ////////////////////
main(){
    srand(time(NULL)) ;
    int number_of_simulation = 0 ; int number_of_mines = -1 ; int write_file
= 1 ;
    int concentration = -1 ;
    cout<<"Difficulty ?? "<<"Init Type ?? "<<"Epoch ?? "<<"Concentration ??
"<<"# mines ?? "<<"To write files ?? "<<endl;
    cin >> difficulty >> init_type >> number_of_simulation >> concentration
>> number_of_mines >> write_file;
    int hint_num = -999  ;
    if(difficulty == 4){
        cout<<"How many hints ? "<<endl;
        cin >> hint_num ;
    }
    double ttl_speed = 0 ;
```

```
    double ttl_stuck = 0 ;
    double ttl_dist = 0 ;
    double ttl_connection = 0 ;
    double max_speed = 0 ;
    double min_speed = 99999 ;
    double max_stuck = 0 ;
    double min_stuck = 99999 ;
    double max_dist = 0 ;
    double min_dist = 99999 ;
    double max_connection = 0 ;
    double min_connection = 99999 ;
    for (int epoch = 0 ; epoch < number_of_simulation ; epoch++) {
        Initialize(number_of_mines , concentration) ;
        int type = max(-1 , hint_num) ;
        if(init_type == 1){
            vector<vector<string> >init_safe = Initial_Safe_Cells(type) ;
            for(int i = 0 ; i < init_safe.size() ; i++){
                KB.push_back(init_safe[i]) ;
            }
        }
        else{
            vector<vector<string> >init_safe =
Initial_Safe_Cells_Random(type) ;
            for(int i = 0 ; i < init_safe.size() ; i++){
                KB.push_back(init_safe[i]) ;
            }
        }
        cout<<"==============="<<endl;
        int stuck = 0 ;
        int iter_cnt = 0 ;
        struct timeval start , end ;
        gettimeofday(&start , 0) ;
        int prev_KBsize = -1  ;
        while(Check_Win() == 0){
            iter_cnt += 1 ;
            int prev_num_marked = num_marked ;
            //cout<<"KB.size() is "<<KB.size()<<" ; mean KB size is :
"<<MeanKBSize()<<endl;
```

```
            if(KB.size() > 15000){
                break ;
            }
            int dilema = GameFlow() ;
            if(dilema == 1 || Check_Logic() == 1 || Check_Statement() == 1){
                cout<<"Wrong Logic !"<<endl;
                break ;
            }
            //cout<<"In "<<iter_cnt<<" 's iteration , Founded Mines : " ;
            //cout<<founded_mines<<endl ;
//            ShowMarked() ;
            //cout<<"==============="<<endl;
            if(num_marked == prev_num_marked){
                stuck += 1 ;
                if(KB.size() == prev_KBsize){
                    stuck += 3 ;
                }
                cout<<"stuck in cnt : "<<stuck<<" , KB size is
"<<KB.size()<<endl;
                if(stuck >= 8){
                    break ;
                }
            }
            else{
                stuck = 0 ;
            }
            prev_KBsize = KB.size() ;
        }
        gettimeofday(&end , 0) ;
        int sec = end.tv_sec - start.tv_sec ;
        int usec = end.tv_usec - start.tv_usec ;
        double elapsedtime = sec + double(0.000001) * double(usec) ;
        cout<<"Elapsed time : "<<elapsedtime<<endl; ;
        cout<<endl ;
        double percentage = 0 ;
        if(founded_mines != mines){
            percentage = double(double(100)*Find_percentage()) ;
            cout<<">> Stuck ......"<<endl;
```

```
            cout<<">> Exploration Percentage : "<<percentage<<" %"<<endl  ;
        }
        else{
            cout<<">> Successfully Explored !"<<endl;
            percentage = double(100) ;
        }
        cout<<">> This inference epoch is correct ? :
"<<Check_correctness()<<endl;
        cout<<endl ;
        ShowInitHint() ;
        Show() ;
        ShowMarked() ;
        double minesavgdist = MinesAvgDistance() ;
        cout<<"Mines' Concentration : "<<minesavgdist<<endl ;
        double initcellconnection = InitCellConnection() ;
        cout<<"Init Cell Connection : "<<initcellconnection<<endl;

        if(write_file == 1){
            stringstream ss;
            ss << epoch+50+1;
            string str = "";
            ss >> str ;
            string filename = "./difficulty/2/" + str + ".txt";
            ofstream ofs;
            ofs.open(filename.c_str());
            if (!ofs.is_open()) {
                cout << "Failed to open file.\n";
                return 1234; // EXIT_FAILURE
            }

            ofs << difficulty <<"\n" ;
            ofs << minesavgdist << "\n";
            ofs << init_type << "\n" ;
            ofs << WhereInitHints.size()<<"\n" ;
            ofs << elapsedtime <<"\n" ;
            int cnt = 0 ;
            for(int i = 0 ; i < size1 ; i++){
                for(int j = 0 ; j < size2 ; j++){
```

```
        if(marked[i][j] != 0){
            cnt += 1 ;
        }
    }
}
ofs << cnt << "/" << size1*size2 <<"\n" ;
ofs << percentage <<"\n" ;
ofs << initcellconnection << "\n";

ofs << "==="<<"\n" ;
for(int i = 0 ; i < size1 ; i++){
    for(int j = 0 ; j < size2 ; j++){
        ofs << Board[i][j]<<" ";
    }
    ofs << "\n" ;
}
ofs << "\n" ;
ofs << "==="<<"\n" ;
for(int i = 0 ; i < size1 ; i++){
    for(int j = 0 ; j < size2 ; j++){
        int flag = 0 ;
        for(int k = 0 ; k < WhereInitHints.size() ; k++){
            int r = WhereInitHints[k].first ;
            int c = WhereInitHints[k].second ;
            if(r == i && c == j){
                flag = 1 ;
                break ;
            }
        }
        if(flag == 0){
            ofs<<"*"<<" ";
        }
        else{
            ofs<<0<<" ";
        }
    }
    ofs <<"\n" ;
}
```

```
        ofs << "\n" ;
        ofs << "==="<<"\n" ;
        for(int i = 0 ; i < size1 ; i++){
            for(int j = 0 ; j < size2 ; j++){
                if(marked[i][j] == 1){
                    ofs<<1<<" ";
                }
                else if(marked[i][j] == 2){
                    ofs<<0<<" ";
                }
                else{
                    ofs<<"*"<<" ";
                }
            }
            ofs<<"\n" ;
        }
        ofs.close();
    }


    ttl_speed += elapsedtime ;
    ttl_stuck += percentage ;
    ttl_dist += minesavgdist ;
    ttl_connection += initcellconnection ;

    max_speed = max(elapsedtime , max_speed) ;
    max_stuck = max(percentage , max_stuck) ;
    max_dist = max(minesavgdist , max_dist) ;
    max_connection = max(initcellconnection , max_connection) ;
    min_speed = min(elapsedtime , min_speed) ;
    min_stuck = min(percentage , min_stuck) ;
    min_dist = min(minesavgdist , min_dist) ;
    min_connection = min(initcellconnection , min_connection) ;
}
if(write_file == 1){
    string filename = "./difficulty/2/avg100.txt";
    ofstream sofs;
    sofs.open(filename.c_str());
```

```
        if (!sofs.is_open()) {
            cout << "Failed to open file.\n";
            return 1234; // EXIT_FAILURE
        }
        sofs<< difficulty <<" "<< init_type <<" "<< number_of_simulation <<"
"<< concentration <<" "<< number_of_mines <<"\n";
        sofs << "AVG speed :
"<<double(double(ttl_speed)/double(number_of_simulation))<<"\n" ;
        sofs << "AVG stuck :
"<<double(double(ttl_stuck)/double(number_of_simulation))<<"\n" ;
        sofs << "AVG dist :
"<<double(double(ttl_dist)/double(number_of_simulation))<<"\n" ;
        sofs << "AVG connection :
"<<double(double(ttl_connection)/double(number_of_simulation))<<"\n" ;
        sofs << "Min speed : "<<min_speed<<"\n" ;
        sofs << "Max speed : "<<max_speed<<"\n" ;
        sofs << "Min stuck : "<<min_stuck<<"\n" ;
        sofs << "Max stuck : "<<max_stuck<<"\n" ;
        sofs << "Min dist : "<<min_dist<<"\n" ;
        sofs << "Max dist : "<<max_dist<<"\n" ;
        sofs << "Min connection : "<<min_connection<<"\n" ;
        sofs << "Max connection : "<<max_connection<<"\n" ;
        sofs.close() ;
    }
    return 0 ;
}
```